Where are we?

The first two blocks of the course dealt with . . .

- Basic notions of agency
- Intelligent problem-solving
- Heuristic search, constraints
- Logic & logical reasoning
- Reasoning about actions and time

In the remainder of the course we will talk about . . .

- Planning
- Uncertainty

What is planning?

- **Planning** is the task of coming up with a sequence of actions that will achieve a goal
- We are only considering **classical planning** in which environments are
  - fully observable (accessible),
  - deterministic,
  - finite,
  - static (up to agents' actions),
  - discrete (in actions, states, objects and events).
- (Lifting some of these assumptions will be the subject of the “uncertainty” part of the course)

Why planning?

- So far we have dealt with two types of agents:
  1. Search-based problem-solving agents
  2. Logical planning agents
- Do these techniques work for solving planning problems?
**Why planning?**

- Consider a search-based problem-solving agent in a robot shopping world.
- Task: Go to the supermarket and get milk, bananas and a cordless drill.
- What would a search-based agent do?

```
Buy Tuna Fish
Buy Arugula
Buy Milk
Go To Supermarket
Go To Class
Buy a Dog
Talk to Parrot
Sit Some More
Read A Book
...```

```
Go To Supermarket
Go To Sleep
Go To School
Go To Pet Store
Etc. Etc. ...
Sit in Chair
Start
Finish
```

**Problems with search**

- No goal-directedness.
- No problem decomposition into sub-goals that build on each other.
  - May undo past achievements
  - May go to the store 3 times!
- Simple goal test doesn’t allow for the identification of milestones.
- How do we find a good heuristic function?
- How do we model the way humans perceive complex goals and the quality of a plan?

**How about logic & deductive inference?**

- Generally a good idea, allows for “opening up” representations of states, actions, goals and plans.
- If \( \text{Goal} = \text{Have}(\text{Bananas}) \land \text{Have}(\text{Milk}) \) this allows achievement of sub-goals (if independent).
- Current state can be described by properties in a compact way (e.g. \( \text{Have}(\text{Drill}) \) stands for hundreds of states).
- Allows for compact description of actions, for example:
  \[
  \text{Object}(x) \Rightarrow \text{Can}(a, \text{Grab}(x))
  \]
- Allows for representing a plan hierarchically, e.g.:
  \[
  \text{GoTo}(\text{Supermarket}) = \text{Leave}(\text{House}) \land \text{ReachLocationOf}(\text{Supermarket}) \land \text{Enter}(\text{Supermarket})
  \]
  then decompose further into sub-plans.

**How about logic & deductive inference?**

- Problems:
  1. In its general form either awkward (propositional logic) or tractability problems (first-order logic), high complexity.
  2. If \( p \) is a sequence that achieves the goal, then so is \( [a, a^{-1}]p \)!

- Solutions: We need
  1. To reduce complexity to allow scaling up.
  2. To allow reasoning to be guided by plan ‘quality’/efficiency.

- Do 1. today; 2. next time.
Representing planning problems

- Need a language expressive enough to cover interesting problems, restrictive enough to allow efficient algorithms.
- **Planning Domain Definition Language** or PDDL
- PDDL will allow you to express:
  1. states
  2. actions: a description of transitions between states
  3. and goals: a (partial) description of a state.

### Representing States and Goals in PDDL

**States** represented as conjunctions of propositional or function-free first order positive literals:
- $Happy \land Sunshine, \ At(Plane_1, Melbourne) \land \ At(Plane_2, Sydney)$

So these aren’t states:
- $At(x, y)$ (no variables allowed),
  $\ Love(Father(Fred), Fred)$ (no function symbols allowed)
- $\neg Happy$ (no negation allowed).

Closed-world assumption!

A **goal** is a partial description of a state, and you can use negation, variables etc. to express that description.
- $\neg Happy, \ At(x, SFO), \ Love(Father(Fred), Fred) \ldots$

### Actions in PDDL

**Action** ($\text{Fly}(p, \text{from}, \text{to})$),

**Precond:** $At(p, \text{from}) \land Plane(p) \land Airport(\text{from}) \land Airport(\text{to})$

**Effect:** $\neg At(p, \text{from}) \land At(p, \text{to})$

- Actually **action schemata**, as they may contain variables
- Action name and parameter list serves to identify the action
- **Precondition:** defines states in which action is executable:
  - Conjunction of positive and negative literals, where all variables must occur in action name.
- **Effect:** defines how literals in the input state get changed (anything not mentioned stays the same).
  - Conjunction of positive and negative literals, with all its variables also in the preconditions.
  - Often positive and negative effects are divided into **add list** and **delete list**.

### The semantics of PDDL: States and their Descriptions

$s \models At(P_1, SFO)$ iff $At(P_1, SFO) \in s$

$s \models \neg At(P_1, SFO)$ iff $At(P_1, SFO) \not\in s$

$s \models \phi(x)$ iff there is a ground term $d$ such that $s \models \phi[x/d]$.

$s \models \phi \land \psi$ iff $s \models \phi$ and $s \models \psi$
The Semantics of PDDL: Applicable Actions

- Any action is **applicable** in any state that satisfies the precondition with an appropriate substitution for parameters.
- Example: State
  \[\text{At}(P_1, Melbourne) \land \text{At}(P_2, Sydney) \land \text{Plane}(P_1) \land \text{Plane}(P_2) \land \text{Airport}(Sydney) \land \text{Airport}(Melbourne) \land \text{Airport}(Heathrow)\]

satisfies

\[\text{At}(p, \text{from}) \land \text{Plane}(p) \land \text{Airport}(\text{from}) \land \text{Airport}(\text{to})\]

with substitution (among others)

\[\{p/P_2, \text{from}/Sydney, \text{to}/Heathrow\}\]

The semantics of PDDL: The Result of an Action

- **Result** of executing action \(a\) in state \(s\) is state \(s'\) with any positive literal \(P\) in \(a\)'s **Effects** added to the state and every negative literal \(\neg P\) removed from it (under the given substitution).
- In our example \(s'\) would be

\[\text{At}(P_1, Melbourne) \land \text{At}(P_2, Heathrow) \land \text{Plane}(P_1) \land \text{Plane}(P_2) \land \text{Airport}(Sydney) \land \text{Airport}(Melbourne) \land \text{Airport}(Heathrow)\]

- “PDDL assumption”: every literal not mentioned in the effect remains unchanged (cf. frame problem)
- **Solution** = action sequence that leads from the initial state to a state that satisfies the goal.

Blocks world example

- Given: A set of cube-shaped blocks sitting on a table
- Can be stacked, but only one on top of the other
- Robot arm can move around blocks (one at a time)
- Goal: to stack blocks in a certain way
- Formalisation in PDDL:
  - \(\text{On}(b, x)\) to denote that block \(b\) is on \(x\) (block/table)
  - \(\text{Move}(b, x, y)\) to indicate action of moving \(b\) from \(x\) to \(y\)
  - Precondition for this action: nothing must be stacked on \(x\): \(\text{Clear}(x)\).

Blocks world example

- Action schema:

  \[\text{Action}(\text{Move}(b, x, y)),\]
  \[\text{PRECOND: } \text{On}(b, x) \land \text{Clear}(b) \land \text{Clear}(y)\]
  \[\text{EFFECT: } \text{On}(b, y) \land \text{Clear}(x) \land \neg \text{On}(b, x) \land \neg \text{Clear}(y)\]

- Problem: when \(x = \text{Table} \) or \(y = \text{Table} \) we infer that the table is clear when we have moved a block from it (not true) and require that table is clear to move something on it (not true)
- Solution: introduce another action

  \[\text{Action}(\text{MoveToTable}(b, x)),\]
  \[\text{PRECOND: } \text{On}(b, x) \land \text{Clear}(b)\]
  \[\text{EFFECT: } \text{On}(b, \text{Table}) \land \text{Clear}(x) \land \neg \text{On}(b, x)\]
Does this Work?

- Interpret $\text{Clear}(b)$ as “there is space on $b$ to hold a block” (thus $\text{Clear(Table)}$ is always true)
- But without further modification, planner can still use $\text{Move}(b, x, \text{Table})$:
  - Needlessly increases search space (not a big problem here, but can be)
- So part of solution is to also add $\text{Block}(b) \land \text{Block}(y)$ to precondition of $\text{Move}$

Summary

- Defined the planning problem
- Discussed problems with search/logic
- Introduced PDDL: a special representation language for planning
- Blocks world example as a famous application domain
- Next time: Algorithms for planning!
  
  **State-Space Search and Partial-Order Planning**